Hindawi
Journal of Sensors
Volume 2019, Article ID 1496208, 15 pages
https://doi.org/10.1155/2019/1496208

Research Article

Efficient Aggregation Processing in the Presence of Duplicately Detected Objects in WSNs

Jun-Ki Min,[1] Raymond T. Ng,[2] and Kyuseok Shim[3]

[1]*School of Computer Science and Engineering, KoreaTech, Republic of Korea*
[2]*Department of Computer Science, University of British Columbia, Canada*
[3]*School of Electrical and Computer Engineering, Seoul National University, Republic of Korea*

Correspondence should be addressed to Jun-Ki Min; jkmin@koreatech.ac.kr

Received 4 July 2018; Revised 7 January 2019; Accepted 17 January 2019; Published 15 May 2019

Academic Editor: Tomasz Wandowski

Wireless sensor networks (WSNs) have received increasing attention in the past decades. Owing to an enhancement of MEMS technology, various types of sensors such as motion detectors, infrared radiation detectors, ultrasonic sensors (sonar), and magnetometers can detect the objects within a certain range. Under such an environment, an object without an identifier can be detected by several sensor nodes. However, existing studies for query processing in WSNs simply assume that the sensing regions of sensors are disjoint. Thus, for query aggregation processing, effective deduplication is vital. In this paper, we propose an approximate but effective aggregate query processing algorithm, called DE-Duplication on the Least Common Ancestor* (abbreviated as DELCA*). In contrast to most existing studies, since we assume that each object does not have a unique identifier, we perform deduplication based on similarity. To recognize the duplicately detected events earlier, we utilize the locality-sensitive hashing (LSH) technique. In addition, since the similarity measures are not generally transitive, we adapt three duplicate semantics. In our experiments, by using a transmission cost model, we demonstrate that our proposed technique is energy-efficient. We also show the accuracy of our proposed technique.

1. Introduction

As a key element for Internet of Things (IoT) applications, wireless sensor networks (WSNs) have received increasing attention in recent years. A sensing circuit measures the parameters from the environment surrounding the sensor and transforms them into an electric signal. Distributed sensors allow us to detect events over a wider area. Thus, multiple sensors are generally placed on fields of interest, and objects there are monitored continuously. A sensor network is a network of spatially distributed autonomous small devices with homogeneous or heterogeneous sensors.

Sensor networks have a broad range of applications in various domains such as environment and habitat monitoring applications [1] that collect meteorological data (e.g., temperature, pressure, and humidity), combat field surveillance applications [2] that track the movement of personnel or detect potentially hazardous chemicals, and disaster management applications that detect forest fire or flood. Since sensor nodes have limited battery power and it is often impractical or hard to replace the batteries of sensor nodes in the environments of its applications, minimizing energy consumption at sensor nodes has been one of the most important objectives. Thus, to prolong the lifetime of a network, much work for in-network query processing has been proposed [3–5].

In the existing literature for WSNs, a common assumption is that the sensing regions of sensors are disjoint. This assumption simplifies the in-network query processing for WSNs. However, due to an enhancement of MEMS technology, various types of sensors such as motion detectors, infrared radiation detectors [1], ultrasonic sensors (sonar), and magnetometers [2] detecting objects within a certain detection range of a sensor become cheap. Thus, it is economically feasible to deploy multiple sensors, with overlapping sensing ranges. When multiple sensor nodes are placed, it is possible that an identical object is detected by several sensors whenever it appears in the overlapping detection area of those sensor nodes. Furthermore, due to some errors resulted from the device noises, calibration errors, and so on, the sensing values of several sensors for an identical object may be different. Such detection of duplicates by several sensor nodes makes the in-network query evaluation very problematic.

To alleviate the drawback of observing multiple sensing values for an identical object, deduplication of the duplicately observed objects by several sensor nodes is critical for aggregate query processing. A brute-force method to solve this problem is to force every sensor node to send its detected objects to the base station so that the base station computes the aggregation result after performing deduplication. However, this scheme would waste a lot of sensor energy unnecessarily, since communication consumes much more power than local computation and memory access [6]. In this paper, we study the effective processing of in-network aggregation queries in sensor networks when the duplicates of an identical object are detected by several sensor nodes. In particular, in contrast to the scheme studied in [7], in which an exact aggregation is computed by utilizing the identifier of each detected object, we propose an approximate but effective in-network aggregation algorithm, called DELCA* which is the abbreviation of DE-Duplication on the Least Common Ancestor*. To the best of our knowledge, this is the first study for approximate in-network aggregation when an identical object can be detected by several sensor nodes and each object does not have a unique identifier.

For habitat monitoring (such as [8]), a small sensor, called a collar, is sometimes attached on each animal, providing an identifier. For such environments, an efficient aggregate query processing technique in WSNs was proposed in [7]. By using the identifier of each object, duplicates can be easily and accurately identified. However, in many other applications, such as military applications with battle fields, an object of interest may not have a prearranged identifier. For instance, the population of enemy agents in a combat field is quite important information to a commander. In order to detect the number of agents, the collaboration of the sensor nodes scattered in the battle field is required. However, in this case, it is hard to assign a unique identifier to each enemy agent. A similar scenario occurs in ocean monitoring of sea mammals, when the mammals are not tagged with collars. Thus, an approximate but effective in-network aggregate query processing is vital. Recall that for an identical object, the sensed attribute values of the object may not be the same due to device noises, calibration errors, and so on. Thus, the similarity measures such as cosine distance and L_p distance can be utilized to identify the duplicates in such an environment. More specifically, in order to recognize the duplicates, since the detected objects do not have identifiers, some attributes of the detected objects are used to form the feature vectors (e.g., location) of the detected objects, and we approximately recognize the duplicates for each identical object based on similarity.

Similar to the related study in [7], our proposed algorithm DELCA* consists of two phases, called the potential duplicate detection phase and the partial in-network aggregation phase, to minimize the cost of transmitting the sensor readings of the duplicate objects to the base station. During the first phase of detecting potential duplicates, collaboration with sensor nodes is required. However, to recognize the duplicates of an identical object, if each node sends and/or receives the sensor readings to/from the other nodes, it tends to waste a lot of energy due to the large volume of sensor readings. Thus, to reduce the communication overhead for the collaboration, sensor nodes send and receive the compact representation of their detected objects. Specifically, we use locality-sensitive hashing (LHS) [9] as a compact representation. Conceptually, each sensor node s_i transmits the compact representation to several neighboring sensor nodes s_p and then finds the potential duplicates among its readings by using the compact representations coming from those other sensor nodes.

After potential duplicates are identified, the partial in-network aggregation phase is performed. During this phase, an aggregation function AGG such as COUNT, MIN, MAX, and AVG is applied to the uniquely detected objects to obtain a partial aggregation result. Then, each sensor node sends the partial aggregation result and the potential duplicates to its *coordinator node* which performs the deduplication and aggregation. Note that, since each node can send a single partial aggregation result instead of uniquely detected objects to its coordinator node, we can reduce the communication overhead. To identify the duplicates among the potential duplicates at the coordinator, a similarity measure between the feature vectors of objects is used. When a subset of the potential duplicates for an identical object has been determined, another aggregation function to the subset is applied to produce a representative value for the identical object, such as taking the average. Then, the aggregation function AGG is applied to the representative values for the subsets of the potential duplicates. The generated partial result is passed to the base station by exploiting the hierarchical property of a routing tree.

2. Preliminaries

In this section, we first present our assumptions for WSNs and the query syntax to be considered for aggregate query processing and then explain how locality-sensitive hashing (LHS) is utilized for recognizing the duplicates of identical objects.

2.1. Aggregate Query Processing in WSNs. The structure of a WSN considered in this paper is the same as that used in [7]. Let us consider a set of stationary sensor nodes $S = \{s_1, s_2, \cdots, s_n\}$. The sensor readings are collected at the base station with unlimited energy supply where the base station serves as an access point to users to process their aggregate queries. Essentially, the sensor readings are collected at the base station by using a tree routing protocol [10]. A pair of nodes capable of bidirectional wireless communication directly is referred to as *neighbors* to each other. The base station has the information regarding the locations of sensor nodes and the tree routing hierarchy among the sensor nodes. Similar to [7], we do not consider the link failures and node failures here. Link failures can be solved with the retransmission protocols or the multipath routing [11]. In addition, to handle node failures, every alive node broadcasts a beacon signal periodically and detects the failure of other nodes that do not send any beacon signal for a long time.

For a sensor node s_i, another sensor node whose sensing region overlaps with that of s_i is called an *oneighbor*, and the least common ancestor of the sensor node s_i and its oneighbors in the routing path is called the *coordinator* of s_i [7]. For instance, consider a simple WSN shown in Figure 1 where the dotted circle of a sensor node denotes its sensing region and the solid line represents the routing path of a given sensor network. The oneighbors of s_4 are $\{s_3, s_5, s_6\}$, and the coordinator of s_4 is s_1. Meanwhile, the coordinator of s_6 is s_4 since s_4 is the least common ancestor of s_6 as well as the oneighbor of s_6. Since the sensing regions of some sensor nodes may overlap each other, an identical object can be observed by several sensor nodes. Thus, deduplication by the collaboration of sensor nodes is necessary. The aggregate query Q that we consider has the syntax (see Figure 2).

In the above SQL-like query, an aggregation function AGG is applied to an attribute a_k (with $1 \le k \le m$) coming from the result of the inner statement of Q. In the inner statement, we introduce the DUPLICATE BY clause to describe the condition to be satisfied for a set of detected objects to be regarded as the duplicates for the same object. By using the expression $sim(S.feature\_vec) \ge \delta$, a user is requesting the query processor to handle duplicates based on a similarity measure $sim(S.feature\_vec)$ where *feature_vec* consisting of some attributes (e.g., {x-positions, y-position}) is the feature vector of each objet and δ is the minimum similarity threshold.

As mentioned previously, due to the sensor noise and calibration errors, each node may observe different values in the attributes of an identical object. Thus, the SELECT clause in the inner statement of Q specifies the representative values of every attribute attr$_i$ for each distinct object by using an aggregation function AGG$_i$ for $1 \le k \le m$. To express the condition of the target objects to apply the aggregation function AGG, we use the WHERE clause in the outer statement. Since the representative attributes a_i of an object are also used for the selection predicate p of the query Q, the result (i.e., $a_1, \cdots, a_m$) of the inner statement must contain the attributes used in the selection predicate p as well as the aggregation attribute a_k.

Table 1 lists the several popular similarity/distance measures between a pair of the feature vectors u and v. Typically, the L_p distance and Hamming distance are used in the literature [9, 12]. However, distance measures such as the L_p and Hamming distances can be easily transformed to a similarity measure with normalization. We define the L_p similarity and the Hamming similarity as follows:

Definition 1. Let u and v be a pair of d-dimensional vectors and the domain of the i-th dimension be $dom(i)$. Then, the maximum L_p distance in a d-dimensional space becomes $(\sum_i |\text{dom }(i)|^p)^{1/p}$, where $|\text{dom }(i)|$ is the length of dom (i). The L_p similarity $sim_{L_p}(u, v)$ of u and v is the closeness of two vectors according to the normalized L_p distance. That is,

$$sim_{L_p}(u, v) = \frac{(\sum_i |\text{dom }(i)|^p)^{1/p} - L_p(u, v)}{(\sum_i |\text{dom }(i)|^p)^{1/p}} = 1 - \frac{L_p(u, v)}{(\sum_i |\text{dom }(i)|^p)^{1/p}}. \quad (1)$$

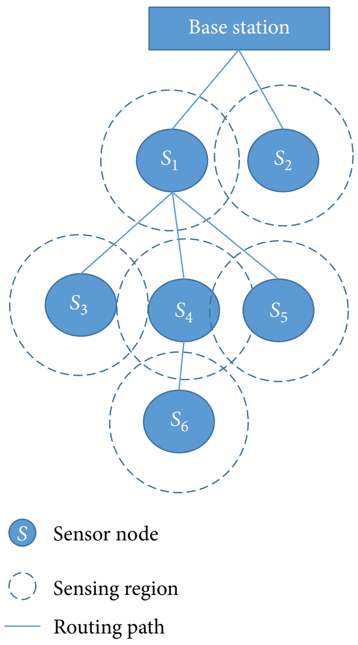


Figure 1: A simple sensor network consisting of 6 sensor nodes.

```
Q: SELECT AGG ($a_k$)
   FROM (
      SELECT AGG$_1$ (S.attr$_1$) as $a_1$, ···, AGG$_m$ (S.attr$_m$) as $a_m$
      FROM Sensor S
      DUPLICATE BY $sim(S.feature\_vec) \ge \delta$,
   )
   WHERE $p(a_1, \cdots, a_m)$
```

Figure 2: A query syntax.

Table 1: Similarity/distance measure.

Similarity	Formula
Cosine similarity	$\cos(\theta(u, v)) = \frac{u \cdot v}{\|u\| \cdot \|v\|}$
Jaccard coefficient	$J(u, v) = \frac{\lvert u \cap v\rvert}{\lvert u \cup v\rvert}$
L_p distance	$L_p(u, v) = \left(\sum_i \lvert u_i - v_i\rvert^p\right)^{1/p}$
Hamming distance	$d_H(u, v) = \lvert(u - v) \cup (v - u)\rvert$

Definition 2. Let u and v be a pair of vectors in a d-dimensional Hamming space. The Hamming similarity $sim_H(u, v)$ of u and v is the fraction of common bits between u and v. In other words,

$$sim_H(u, v) = 1 - \frac{d_H(u, v)}{d}. \quad (2)$$

For brevity, we simply refer to the similarity between the feature vectors of a pair of objects as the similarity between the pairs of objects. Furthermore, when the feature vectors of a pair of objects are similar according to a similarity measure, we simply say that the pairs of objects are similar.

2.2. Locality-Sensitive Hashing (LSH). In this section, we briefly present the locality-sensitive hashing (LSH).

Definition 3. A locality-sensitive hashing scheme is a distribution on a family $\mathscr{H}$ of hash functions operating on a collection of objects, such that for two objects x and y,

$$P_{h\in\mathscr{H}}(h(x) = h(y)) = sim(x, y), \quad (3)$$

where $sim(x, y)$ is a similarity between x and y.

Given a hash function family $\mathscr{H}$ that satisfies equation (3), we say that $\mathscr{H}$ is a locality-sensitive hash function family corresponding to the similarity measure. Note that the more similar a pair of objects is, the higher the collision (or matching) probability is. Definition 3 is slightly different from the original definition in [9]. However, they are fundamentally the same and many LSH functions satisfy this definition [13]. LSH families have been developed for several similarity measures in the past [9, 13, 14].

It is convenient to have a hash function family that maps objects to $\{0, 1\}$ (i.e., a single bit number) since a k-bit hash value for an object is the concatenation of the k different hash functions. We refer to such a k-bit hash value as a *LSH vector*. The similarity between a pair of objects u and v is estimated by counting the number of matching coordinates in their corresponding LSH vectors. Actually, we can always find such a binary hash function family with a slight change for any similarity measure whose locality-sensitive hash function family exists, as the following lemma states:

Lemma 1 (see [13]). *Given a LSH function family $\mathscr{H}$ corresponding to $sim(u, v)$, we can obtain a LSH function family $\mathscr{H}'$ that maps objects to $\{0, 1\}$ and corresponds to the similarity function $(1 + sim(u, v))/2$.*

For instance, although a LSH function for the L_p distance [14] does not generate a single bit number, we can get a binary hash function family for the L_p distance by Lemma 1.

3. Related Work

As one of the most important elements for the IoT platform, WSN is used in a broad class of IoT applications. In the database community, several sensor data management systems such as Cougar [3] and TinyDB [10] have been introduced. One of the well-known approaches to reduce the energy drain of sensor networks is the in-network aggregation. Tiny AGgregation (abbreviated as TAG) proposed by Madden et al. in [5] is the first study for in-network aggregation to decrease the battery usage by reducing the communication overheads. In TAG, the partial aggregation values are computed incrementally with routing up from the leaf nodes to the base station in the routing tree.

Several approximate in-network aggregation techniques have also been developed [11, 15–19]. Considine et al. [15] proposed a robust method by using a Flajolet-Martin sketch (abbreviated to FM sketch) and the multipath routing for processing approximate aggregate queries in the presence of failures. Shrivastava et al. [16] presented the q-digest structure to approximately compute quantile queries. Nath et al. [11] introduced a general framework based on synopsis diffusion for various approximate aggregate queries. For aggregation functions such as MIN and MAX, Silberstein et al. [17] developed an algorithm with the goal of minimizing the communication cost in a sensor network while guaranteeing the accuracy of aggregate query results. Xu et al. [19] proposed an approximated aggregation technique using data compression to reduce the amount of transmitted data. Recently, Xiao et al. [18] presented an energy-efficient aggregation technique for an unreliable network based on a probabilistic network model. Some aggregation techniques for a spatial region in sensor networks have also been proposed [20–22]. Soheili et al. [21] proposed a distributed spatial index, called SPIX, to compute an aggregate value for a user-defined spatial region. Zhuang and Chen [22] introduced a max regional aggregate query to find a region with the maximum aggregation value. In [20], Choi and Chung presented a technique using the smallest enclosing circles to process a max regional aggregate query.

All the studies mentioned so far do not allow the sensing regions to overlap and thus cannot handle the detected duplicates of an identical object. Due to the recent enhancement of MEMS technology, some types of sensors detecting the objects within a certain range become economical. For instance, in [23], an effective technique for actuating the camera sensors in a certain region collaborated with other types of sensors, called scalar sensors. Thus, if the sensor readings measured by the scalar sensors for a certain region exceed the predefine threshold, the camera sensors for the region are activated. The most relevant work to ours is [7] in which an exact in-network aggregation method, called LCA EA, was proposed for the case of when sensor nodes detect an object simultaneously. Although LCA EA consists of two phases like ours, since the basic assumption is that each object has a unique identifier, the exact aggregation result is computed by using the identifiers of the detected objects. However, for many applications, it is hard to assign a unique identifier to each object in practice. Thus, we propose an approximate but effective in-network aggregation algorithm for the case of when an identical object can be detected by several sensor nodes and each object does not have a unique identifier.

4. Effective Aggregation in the Presence of Duplicately Detected Objects

4.1. An Overview of the Proposed DELCA* Algorithm. If each node can effectively identify the unique objects among its detected objects, since each node can send the partial aggregation result generated from the unique objects and the (potential) duplicate objects instead of sending all of the detected objects to its coordinator, the communication overhead can be reduced significantly. To achieve this goal, in this paper, we propose the DELCA* algorithm.

In our DELCA* algorithm, the processing of an aggregate query Q is split into two phases: the potential duplicate detection phase and the partial innetwork aggregation phase.

(i) Potential duplicate detection phase: during this phase, each node identifies the potential duplicates among its detected objects. To recognize the potential duplicates, each sensor node s_i requires the information of the objects detected by its oneighbors whose sensing regions overlap with that of s_i. Thus, each node first broadcasts the compact representation (i.e., LSH vectors) of its detected objects' feature vectors to its oneighbors. Then, after receiving the compact representations from its oneighbors, each node splits the detected objects into the potential duplicate partition and unique object partition

(ii) Partial in-network aggregation phase: each node computes a partial aggregation result with the unique objects satisfying the selection predicates p of a given query Q. The partial aggregation results and the potential duplicates are continually routed up from the leaf nodes to the base station. When a node receives the partial aggregation results from its child nodes, it updates its partial aggregation result with those of the child nodes. It also merges its potential duplicates with the received potential duplicates. When the duplicates of an identical object are collected at the coordinator, it computes the representative value of the duplicates with respect to the SELECT clause of the inner SQL statement of the query Q. Then, the duplicates of an identical object are removed from the potential duplicate objects. In addition, the node updates the aggregation result with the aggregation attribute values of the representatives satisfying the selection predicate p. Finally, the node sends the updated partial aggregation result and the remaining potential duplicates to its parent node in the routing path

Figure 3 illustrates the behavior of our proposed DELCA* algorithm. Let us assume that the sensor nodes s_3 and s_4 detect an object u as u' and u'', respectively, because u appears in the overlapped area of the sensing regions of s_3 and s_4 as illustrated in Figure 3(a). Moreover, the sensing values of the same object detected by several sensors can be different. Meanwhile, another object v is detected by s_4 only, since v is located within the nonoverlapped sensing region of s_4 as illustrated in Figure 3(a).

During the potential duplicate detection phase of DELCA*, s_4 broadcasts the set of LSH vectors $\{l_{u''}, l_v\}$ to its oneighbors s_3, s_5, and s_6. Similarly, s_3 broadcasts $\{l_{u'}\}$ to its oneighbor s_4 as shown in Figure 3(b) where $l_{u'}$, $l_{u''}$, and l_v are the LSH vectors for the u', u'', and v, respectively. Note that owing to the nature of LSH, if the pairs of objects are similar, their LSH vectors tend to be similar too. Thus, by using the received LSH vectors $\{l_{u''}, l_v\}$, s_3 could determine that u' is a potential duplicate object since $l_{u''}$ from s_4 tends to be similar to $l_{u'}$ in s_3. Similarly, s_4 decides that u'' is a potential duplicate and v is a unique object.

In the partial in-network aggregation phase, s_4 computes the partial aggregation result *ParAgg* using the unique object v if v satisfies the selection predicate p of the query Q. Then, as illustrated in Figure 3(c), s_4 sends the partial aggregation result *ParAgg* and the potential duplicate u'' to its parent s_1. In contrast, s_3 has duplicate object u' only; s_3 just sends u' to s_1.

Since s_1 is the coordinator of s_4 and s_3, s_1 first calculates the representative of duplicates u' and u'' and updates the partial aggregation result *ParAgg* using the computed representatives. Then, the updated aggregation result *ParAgg* is transmitted to the base station as shown in Figure 3(d).

In Sections 4.2 and 4.3, we present the details of the potential duplicate detection phase and the partial in-network aggregation phase, respectively.

4.2. Potential Duplicate Detection Phase. During this phase, each sensor node distinguishes unique objects and potential duplicates from its detected objects by the collaboration with its oneighbors. The pseudocode of the first phase is presented in Figure 4.

Since the detected objects do not have any identifiers, duplicity is determined by the similarity of the feature vectors of the detected objects. As addressed in Section 2.2, the similarity between a pair of objects is estimated by utilizing their corresponding k-bit LSH vectors. Thus, each sensor node generates a set of LSH vectors *LSHvects* for the set of its detected objects E (lines 1-4 in Figure 4). Let $h_i()$ be a hash function in a locality-sensitive hash function family $\mathscr{H}$ satisfying equation (3) that maps a vector to $\{0, 1\}$ and *u.feature_vec* be the feature vector of an object u. Each sensor node generates a k-bit LSH vector l_u for each object u in E by concatenation of k hash functions $h_1(), \cdots, h_k() \in \mathscr{H}$ (line 3 in Figure 4) and each generated k-bit LSH vector l_u is added into *LSHvects* (line 4). Then, each sensor node s_i broadcasts a set of LSH vectors $LSHvects = \{l_1, l_2, \cdots, l_{|E|}\}$ to its oneighbors (line 5). At the same time, s_i also receives the multiple k-bit LSH vector sets from its oneighbors (line 6). Then, s_i identifies the potential duplicates among the objects in E by using the set of k-bit LSH vector sets coming from its oneighbors, denoted as $\mathbb{L}$.

We now explain how to compute the similarity of two objects by using their corresponding LSH vectors in order to recognize the duplicates. The similarity of two objects u and v can be estimated by counting the number of common

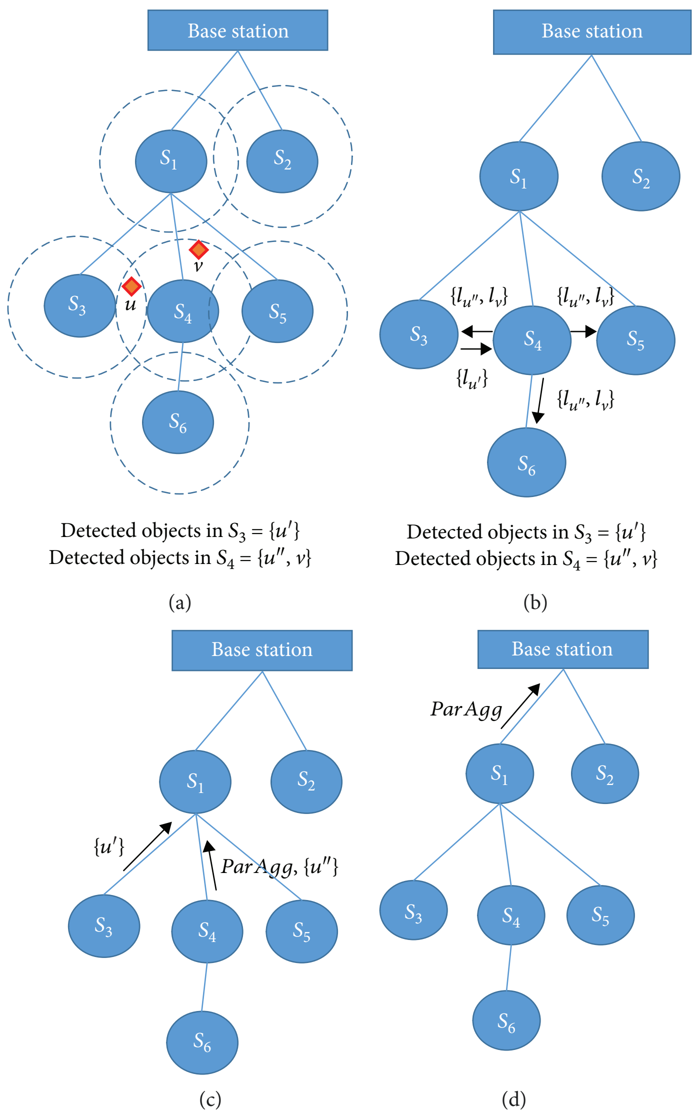

Base station
S_1
S_2
S_3
S_4
S_5
S_6
u
v
Detected objects in $S_3 = \{u'\}$
Detected objects in $S_4 = \{u'', v\}$
(a)
$\{l_{u''}, l_v\}$
$\{l_{u'}\}$
Detected objects in $S_3 = \{u'\}$
Detected objects in $S_4 = \{u'', v\}$
(b)
$\{u'\}$
$ParAgg, \{u''\}$
(c)
$ParAgg$
(d)

FIGURE 3: The behavior of DELCA*.

```
Procedure DELCA*_1ST ($\delta$, $E$)
$\delta$: user defined threshold, $E$: a set of detected objects at this node
begin
  (i)     $LSHvects = \emptyset$
  (ii)    for each object $u \in E$ do
  (iii)   $l_u = h_1(u.feature\_vec) \cdot h_2(u.feature\_vec) \cdot \cdots \cdot h_k(u.feature\_vec)$
              where $h_1$ (with $1 \le i \le$ k) is in $H$ satisfying Eq. (2)
  (iv)    add $l_u$ to $LSHvects$
  (v)     Send $LSHvects$ to the oneighbors of this node
  (vi)    Let L be $\{LSHvects_1, LSHvectsi_2, \ldots, LSHvects_m\}$ where $LSHvects$, is the LSH
           vectors coming from the $i$-th oneighor and $m$ is the number of its oneighbors
  (vii)   for each LSH vectore $l_u \in LSHvects$ do
  (viii)      for each LSH vectore $l_u \in LSHvects$, in L do
  (ix)            if $k - d_H(l_u, l_v) \ge [k \cdot \delta]$ (or $[k \cdot \frac{1+\delta}{2}]$ for sim$_{L_p}$)
  (x)                 Add $u$ to potential_duplicates
  (xi)    $unique\_objects = E\text{-}potential\_duplicates$
end
```

FIGURE 4: The procedure for the potential duplicate detection phase in DELCA*.

bits in two k-bit LSH vectors l_u and l_v. The number of common bits between l_u and l_v can be expressed by $k - d_H(l_u, l_v)$ where $d_H(l_u, l_v)$ is the Hamming distance between l_u and l_v.

Let us assume that a user uses the Hamming similarity sim_H of Definition 2 to recognize the duplicates. Then, the LSH function family $\mathscr{H}_H$ presented in [9] is applied to produce a k-bit LSH vector for each object. The probability that the hash values of two objects u and v are the same is equal to the similarity of u and v (i.e., $Pr_{h\in\mathscr{H}_H}[h(u) = h(v)] = sim_H(u, v)$). Thus, with a minimum similarity threshold δ_H, if the number of common bits of l_u and l_v (i.e., $k - d_H(l_u, l_v)$) is at least $\lfloor k \cdot \delta_H \rfloor$, we regard that the two objects u and v are similar, and thus, they become potential duplicates. For instance, let 8-bit LSH vectors l_u and l_v for objects u and v be 00000000 and 01000100, respectively. Then, $d_H(l_u, l_v)$ is 2. When δ_H is 0.7, we regard that u and v are duplicates since $k - d_H(l_u, l_v)$ $(= 6(= 8 - 2))$ is greater than $\lfloor k \cdot \delta_H \rfloor$ $(= 5(= \lfloor 8 \cdot 0.7 \rfloor))$.

Since locality-sensitive hash function families for L_1 similarity [12] and cosine similarity [13] also map objects to $\{0, 1\}$ (i.e., a single bit number), similar arguments hold on a minimum threshold δ_{L_1} on L_1 similarity sim_{L_1} and a minimum threshold δ_θ on cosine similarity.

Given a minimum threshold δ_{L_p} for the similarity sim_{L_p} by Definition 1, the LSH function family $\mathscr{H}_{Lp}$ presented in [14] is applied. Unlike the other hash families, a hash function in $\mathscr{H}_{Lp}$ generates a scalar value rather than $\{0, 1\}$. However, based on Lemma 1, we can identify the potential duplicates when the L_p similarity sim_{L_p} is used too. Given a pair of two objects u and v, let a pair of k-bit LSH vectors l_u and l_v be generated by using a hash function family Λ which operates on the domain of the functions in $\mathscr{H}_{Lp}$ for L_p similarity and maps the elements in the domain to $\{0, 1\}$. Then, if the number of common bits for l_u and l_v is at least $\lfloor k \cdot ((1 + \delta_{L_p})/2) \rfloor$, we can regard that $sim_{L_p}(u, v)$ is at least δ_{L_p}. By Lemma 1, we can obtain a binary hash function family and the corresponding similarity measure of two objects u and v is changed to $sim'(u, v) = (1 + sim_{L_p}(u, v))/2$.

Thus, if $sim_{L_p}(u, v) \geq \delta_{L_p}$, we have $sim'(u, v) \geq ((1 + \delta_{L_p})/2)$ since $sim_{L_p}(u, v) = 2sim'(u, v) - 1 \geq \delta_{L_p}$. In addition, by the k stochastic repetitions, the similarity is estimated as $sim'(u, v) = 1 - (d_H(l_u, l_v))/k$. Thus, if the number of common bits (i.e., $k - d_H(l_u, l_v)$) between l_u and l_v is at least $\lfloor k \cdot ((1 + \delta_{L_p})/2) \rfloor$, we obtain $sim_{L_p}(u, v) \geq \delta_{L_p}$.

As explained above, for a LSH vector l_u in *LSHvects* of a sensor node s_i, if there is a similar LSH vector $l_v \in LSHvects$ in $\mathbb{L}$ such that $k - d_H(l_u, l_v) \geq \lfloor k \cdot \delta \rfloor$ (or $\lfloor k \cdot ((1 + \delta)/2) \rfloor$ for sim_{L_p}), the corresponding object u is inserted into *potential_duplicates* (lines 4-7 in Figure 4). After obtaining the potential duplicates, each sensor can extract the set of uniquely detected objects from E and insert it into *unique_objects* (line 8).

Let the total number of detected objects be n. Then, for a sensor node s_i, since the number of the detected objects at s_i is at most n and we can compute a LSH vector for each object with $O(1)$ time by using hash functions, *LSHvects* is constructed in $O(n)$ time as well as the size of *LSHvects* is $O(n)$. In addition, the number of the LSH vectors in $\mathbb{L}$ received from its oneighbors is also $O(n)$. Then, for each l_u in *LSHvects*, we check whether there is a similar LSH vector l_v in $\mathbb{L}$. Thus, the time complexity of this phase becomes $O(n^2)$.

In the second phase, potential duplicates are sent to coordinators so that further deduplication can take place. Meanwhile, for unique objects, early aggregation can be performed.

4.3. Partial In-Network Aggregation Phase. During the second phase, each node computes a partial aggregation result with the uniquely detected objects satisfying a selection predicate p of a given query Q. Partial aggregation results are gradually merged along the routing path to the base station. By a definition of the coordinator, the duplicates of an identical object detected by oneighbors of s_i are collected at s_i's coordinator. Thus, when a coordinator of s_i receives the potential duplicates from s_i and s_i's oneighbors, the coordinator computes the representative value of each attributes by using the potential duplicates of itself as well as its oneighbors. Then, the duplicates for an identical object are removed to avoid the redundant reflection of duplicates. If the representative satisfies the selection predicate p, the aggregation attribute value of the representative of each detected object is reflected into the partial aggregation result.

The pseudocode of the second phase is presented in Figure 5. Each sensor node first computes the partial aggregation result *ParAgg* with the partial aggregation results $\mathbb{P}$ coming from its child nodes (lines 1-2 in Figure 5) and updates *ParAgg* using *unique objects* (lines 3-4). Then, each node accumulates the potential duplicates sent from its child nodes into $\mathbb{D}$. The potential duplicates of its own is also kept in $\mathbb{D}$ (lines 5-6). Then, the node extracts a set D of the duplicates for an identical object by invoking the function chooseDup() according to the DUPLICATE BY clause in Q and removes D from $\mathbb{D}$ (line 9). We will explain the details of chooseDup() in Section 4.4.

If there is at least a single object $e \in D$ such that this node is the coordinator of the sensor node s detecting e, D is a set of multiple detected duplicates of an object appearing in the overlapped sensing regions of s and s's oneighbors. Thus, in this case, the node first calculates the representative *rep* of D by applying the aggregate functions AGG_i to the attributes a_i (with $1 \leq k \leq m$) of all objects in D. Then, the partial aggregation result *ParAgg* is updated if *rep* satisfies the predicate p of the query Q (lines 10-12). If D does have such an object, we add D into *PDup* to be sent to its parent node for later investigation (line 13). Finally, the partial aggregation result *ParAgg* and the potential duplicates *PDup* are sent to the parent node (line 14).

During the second phase, as shown in Figure 5, each sensor node first computes the partial aggregation result *ParAgg* with $\mathbb{P}$ coming from its child nodes and *unique_objects*. Let us assume that the total number of detected objects is n and the number of child nodes of each sensor is much smaller than n. Then, the time complexity for computing the partial aggregation result *ParAgg* with $\mathbb{P}$ and *unique_objects* is $O(n)$ since $|unique\_objects|$ is at most n. Then, we

```
Procedure DELCA*_2ND ($\delta$)
$\delta$: user defined threshold
begin
  (i)     Let P be the set of partial aggregation values coming from its child nodes.
  (ii)    $ParAgg$ = AGG ($\mathbb{P}$)
  (iii)   for each object $e \in$ unique_objects do
  (iv)        if $p$ ($e$) = TRUE then $ParAgg$ = Agg ($PagAgg$,$e$)
  (v)     Let $\mathbb{D}$ be the set of potential duplicates coming from its child nodes $s_i$
  (vi)    $\mathbb{D} = \mathbb{D} \cup$ $potentia\_duplicates$
  (vii)   $PDup = \emptyset$
  (viii)  while $\mathbb{D} \neq \emptyset$ do
  (ix)        $D$ = chooseDup ($\mathbb{D}$), $\mathbb{D} = \mathbb{D} - D$
  (x)         if this node is the coordinator of a sensor node detecting $e \in D$ then
  (xi)            $rep$ = {AGG$_1$ ($D$.attr$_1$),. . . , AGG$_m$ ($D$.attr$_m$)}
  (xii)           if $p(rep)$ = TRUE then $ParAgg$ = AGG ($ParAgg$,$rep$)
  (xiii)      else $PDup = PDup \cup D$
  (xiv)   send $ParAgg$ and $PDup$ to its parent node
end
```

FIGURE 5: The procedure for the partial in-network aggregation phase in DELCA*.

extract the duplicates D from an identical object from the potential duplicates $\mathbb{D}$ by invoking chooseDup() iteratively and then we update *ParAgg* with D. Since $|\mathbb{D}|$ is at most n, chooseDup() is invoked at most n times. In addition, chooseDup() evaluates the similarities of $O(n \cdot (n-1)/2)$ object pairs in $\mathbb{D}$. Thus, for the worst case, the time complexity of this phase becomes $O(n^3)$. However, the number of detected objects at each sensor as well as |*unique_objects*| and $|\mathbb{D}|$ is much smaller than n since there are multiple sensors in a sensing field. Thus, the second phase does not take much time in practice.

4.4. Semantics of Duplicates. In this section, we present the details of the function chooseDup() used in the procedure DELCA*_2ND shown in Figure 5. Note that many similarity measures such as Hamming similarity, L_p similarity, and cosine similarity are not transitive. For instance, when a Hamming similarity threshold δ_H is 5/6, we can say that "mother" and "mather" are similar since the Hamming distance between them is 1 and the lengths of them are 6. Similarly, "mather" and "father" are also similar. However, "mother" and "father" are not similar with $\delta_H = 5/6$. In this case, we have to decide whether "mother," "mather," and "father" belong to an equivalence class or not since "mather" can bridge the gap from "mother" to "father." To solve this problem, the previous work [24] ignores the nontransitivity of the similarity measures. However, inspired by the studies in [25, 26], we consider three duplicate semantics for the equivalence classes: *weak semantic*, *strict sematic*, and *monoid semantic*. Thus, we allow that each user can choose a semantic out of the three duplicate semantics by declaring with DUPLICATE BY $sim(feature\_vec) \geq \delta$ [WEAK| STRICT|MONOID].

In [26], when a feature vector u is similar to another feature vector v and v is similar to the other feature vector w, although u is not similar to w, the three feature vectors u, v, and w are regarded as the members of the same equivalence class, since v bridges the gap from u to w. We call such a policy the *weak semantic* policy. In [25], the objects whose feature vectors satisfy the transitivity of similarity are considered duplicates. We call this policy the *strict semantic* policy.

Depending on the application or even an individual user, the above two semantics could be too weak or too strong to identify the duplicates. Thus, to handle the nontransitivity of a similarity measure, we offer a hybrid policy called the *monoid semantic* policy. In monoid semantic policy, every member in an equivalent class is similar to the monoid vector of the equivalent class where the monoid vector consists of the mean value of the equivalent class in each dimension.

The pseudocode of chooseDup() is presented in Figure 6. To begin, a pair of objects $\{e_i, e_j\}$ whose feature vectors are the most similar in $\mathbb{D}$ is chosen as an equivalent class D (lines 1-6). Since we initially assign the user-provided threshold δ to *maxsim* (line 1), the most similar pair whose similarity measure is at least δ is selected (lines 4-6). If such a pair does not exist, $\mathbb{D}$ does not have any duplicate object. Thus, we simply return an object in $\mathbb{D}$ (line 7). If there is such a pair, one of the three duplicate semantics is applied (lines 8-26). The lines 8-12 are for the weak semantic policy, the lines 13-17 are for the strict semantic policy, and the lines 18-26 are for the monoid semantic policy.

With respect to the *weak semantic* policy, for an object e_k in $\mathbb{D}$ but not in D, if e_k.*feature_vec* is similar to at least an element in D, e_k becomes a member of D (lines 10-11). Then, D is returned (line 12). To process the *strict semantic* policy, if an object e_k in $\mathbb{D}$ but not in D is similar to all elements in D, e_k becomes a member of D (lines 14-16). By using this mechanism, we produce the equivalent classes each of which satisfies the strict semantic policy. To handle the *monoid semantic* policy in chooseDup, the monoid vector $monoid_{vec}$ for the most similar pair is computed (line 19). For an object e_k whose feature vector is similar to $monoid_{vec}$ (line 20), chooseDup checks whether e_k can be a member of D or not. By the monoid sematic policy, every object in D should be similar to $monoid_{vec}$. Thus, we first compute a temporary mean vector $temp_{vec}$ (line 21). If every object e in $D \cup e_k$ is similar to $temp_{vec}$, e_k becomes a member

```
Procedure chooseDup ($\mathbb{D}$)
begin
  (i)       $maxsim = \delta$
  (ii)      for each $e_i \in \mathbb{D}$ do
  (iii)         for each $e_j\ (\neq e_i) \in \mathbb{D}$ do
  (iv)              if $sim\ (e_i.feature\_vec, e_j.feature\_vec) \geq maxsim$ then
  (v)                   $D = \{e_i, e_j\}$
  (vi)                  $maxsim = sim\ (e_i.feature\_vec, e_j.feature\_vec)$
  (vii)     if $D = \emptyset$ then return $\{e_i\}$
  (viii)    case WEAK
  (ix)          for each $e_k \in \mathbb{D}$ and ek $\notin D$ do
  (x)               if $\exists\ e \in D$ s.t., $sim\ (e.feature\_vec, e_k.feature\_vev) \geq \delta$ then
  (xi)                  $D = D \cup \{e_k\}$
  (xii)         return $D$
  (xiii)    case STRICT
  (xiv)         for each $e_k \in \mathbb{D}$ and $e_k \notin D$ do
  (xv)              if $\forall\ e \in D$, $sim\ (e.feature\_vec, e_k.feature\_vev) \geq \delta$ then
  (xvi)                 $D = D \cup \{e_k\}$
  (xvii)        return $D$
  (xviii)   case MONOID
  (xix)         $monoid_{vec}$ = mean ($\{e_i.feature\_vec, e_j.feature\_vec\}$)
  (xx)          for each $e_k$ in $\mathbb{D}$ and $e_k \notin D$ where $sim\ (monoid_{vec}, e_k.feature\_vec) \geq \delta$ do
  (xxi)             $temp_{vec}$ = mean ($D \cup e_k.feature\_vec$)
  (xxii)            if $\forall\ e \in D \cup \{e_k\}$, $sim\ (e.feature\_vec, temp_{vec}) \geq \delta$ then
  (xxiii)               $D = D \cup \{e_k\}$
  (xxiv)                $monoid_{vec} = temp_{vec}$
  (xxv)         return $D$
end
```

Figure 6: The procedure chooseDup().

of D and $monoid_{vec}$ is adjusted with e_k (lines 23-24). Finally, D is returned (line 25). By using D, a coordinator produces a representative aggregation value for duplicates.

5. Experiments

In this section, we demonstrate the efficiency of our proposed DELCA* algorithm. We implemented three aggregate query processing algorithms: brute-force (BF) method, DELCA*, and LCA*. In the brute-force (BF) algorithm, each sensor transmits its readings to the base station and aggregation is processed at the base station. While DELCA* is the two-phase algorithm introduced in Section 4, LCA* is a single-phase algorithm that is exactly the same as the second phase of DELCA*. Thus, in LCA*, each sensor node simply sends its detected objects to its coordinator since LCA* does not distinguish unique objects and potential duplicates. In other words, in LCA*, all sensor readings are regarded as the potential duplicates and sent to the parent nodes. To compare the performance of implemented methods, we measure the transmission cost of each method based on the cost model presented in [27].

5.1. Test Environment. In our experiment, we place the sensors based on two strategies: the grid placement strategy and the random placement strategy. In the grid placement strategy, the sensing field was divided into equisized grids and a sensor node is placed in the center of each grid. In the random placement strategy, sensor nodes were randomly scattered at the sensing field following the uniform distribution. To deliver the aggregation results to a user, at least a single sensor node has to be reachable to the base station. Thus, in both sensor placement strategies, the base station was located at the center of the sensing field. The routing tree was constructed by using the FHF (First-Heard-Form) algorithm [10].

Table 2: Parameters.

Parameter	Default value
Sensor placement	Grid or random
Size of sensing field	$(1{,}000\ \mathrm{m})^2$
Number of sensor nodes	1024
Communication distance	50 m
Radius of sensing region (s)	20 m
Packet size (p)	48 *bytes*
Number of objects (a)	10,000

Based on a probabilistic model developed in [7] for animals' movement that was calculated from a real data set containing trajectories of Kruger Buffalos from Movebank [28], we generated a synthetic data set to show the effectiveness of our work. In the generated data set, there are three 4-byte artificial attributes: body temperature, weight, and height. The data value distribution of each data set is followed by the distribution in [7]. However, in contrast to that of [7], since there is no identifier annotated with each object in our problem setting, we used the 8-byte position (2-dimensional point) of a detected object as its feature vector. To reflect the measurement errors of all the attributes including feature vectors, we make the measured attribute values by adding a

Name	Definition
Q1	SELECT COUNT (avg_t)
	FROM (
	SELECT AVG (body_temperature) AS avg_t
	FROM ANIMAL
	DUPLICATE BY sim_{L_2} (position) $\geq \delta$ [WEAK|STRICT|MONOID]
	)
	WHERE avg_t $\geq$ 38
Q2	SELECT MAX (min_w)
	FROM (
	SELECT MIN (weight) AS min_w, AVG (height) as avg_h
	FROM ANIMAL
	DUPLICATE BY sim_{L_2} (position) $\geq \delta$ [WEAK|STRICT|MONOID]
	)
	WHERE avg_t $\leq$ 38
Q3	SELECT MAX (min_t)
	FROM (
	SELECT MIN (body_temperature) AS min_t, MAX (weight) as max_w,
	MAX (height) as max_h
	FROM ANIMAL
	DUPLICATE BY sim_{L_2} (position) $\geq \delta$ [WEAK|STRICT|MONOID]
	)
	WHERE avg_t $\leq$ 38
Q4	SELECT R.pos R.min_b
	FROM (
	SELECT MAX (min_t) as max_b
	FROM (
	SELECT MIN (body_temperature) AS min_t
	FROM ANIMAL
	DUPLICATE BY sim_{L_2} (position) $\geq \delta$ [WEAK|STRICT|MONOID]
	)AS S,
	(SELECT AVG (position) as pos, MIN (body_temperature) as min_b
	FROM ANIMAL
	DUPLICATE BY sim_{L_2} (position) $\geq \delta$ [WEAK|STRICT|MONOID]
	) AS R
	WHERE R.min_b = S.max_b

FIGURE 7: Query set.

random noise within the interval $[-1, 1]$ following the uniform distribution to actual values with the probability of 0.01.

With varying the radius of a sensing region, the number of duplicately detected objects is changed. In the grid sensor placement, the minimum distance between two sensor nodes is 31.25 m (=1000 m/32) since there are 1024 (=32^2) sensors at $(1{,}000\,\text{m})^2$-sized sensing field. Thus, the radius of the sensing region s is greater than 31.25 m, and every object is duplicately detected by at least two sensor nodes. In contrast, when s is less than 15.625 m (=31.25 m/2), sensing regions of sensors do not overlap, and thus, there are no duplicately detected objects. As the radius of sensing region increases, the overlapped sensing region increases, and thus, the number of duplicate objects increases. In our experiments, we varied s from 15 m to 30 m. The other parameters used in our experiment are summarized in Table 2.

Similar to many related studies as in [7, 10, 11, 20, 22], we use the transmission cost to estimate the energy consumption of sensor nodes. As addressed in [27], radio wave propagation is highly variable and difficult to model. Thus, we adopt the simplified free space channel model (c^2 power loss) presented in [27], where c is the distance between two sensor nodes. Under this model, to transmit and receive an l-bit packet for distance c, a sensor spends $E_T(l, c)$ and $ER(l)$ units of energy, respectively, which are defined as

$$\begin{aligned} E_T(l, c) &= l \cdot E_{\text{elec}} + \xi_{\text{amp}} \cdot l \cdot c^2, \\ E_R(l) &= l \cdot E_{\text{elec}}, \end{aligned} \tag{4}$$

where E_{elec} represents the energy consumption for running the transmitter or receiver and ξ_{amp} denotes the energy consumption for a transmit amplifier. In this experiment, we set 50 nJ/bit to the electronic circuit constant (E_{elec}) and 100 pJ/bit/m^2 to the transmit amplifier constant (ξ_{amp}), as in [7, 10, 11, 20]. Furthermore, in our experiments, we do not consider retransmission caused by link failure and so on in order to measure the bare transmission cost of each algorithm.

In Figure 7, we report the aggregate queries used in our experiments. Each query is similar to one in [7] except introducing the DUPLICATED BY clause containing the similarity measure sim_{L_2}. Query Q1 is a simple aggregate query which computes the number of animals having high body temperatures. Query Q2 asks the maximum weight of

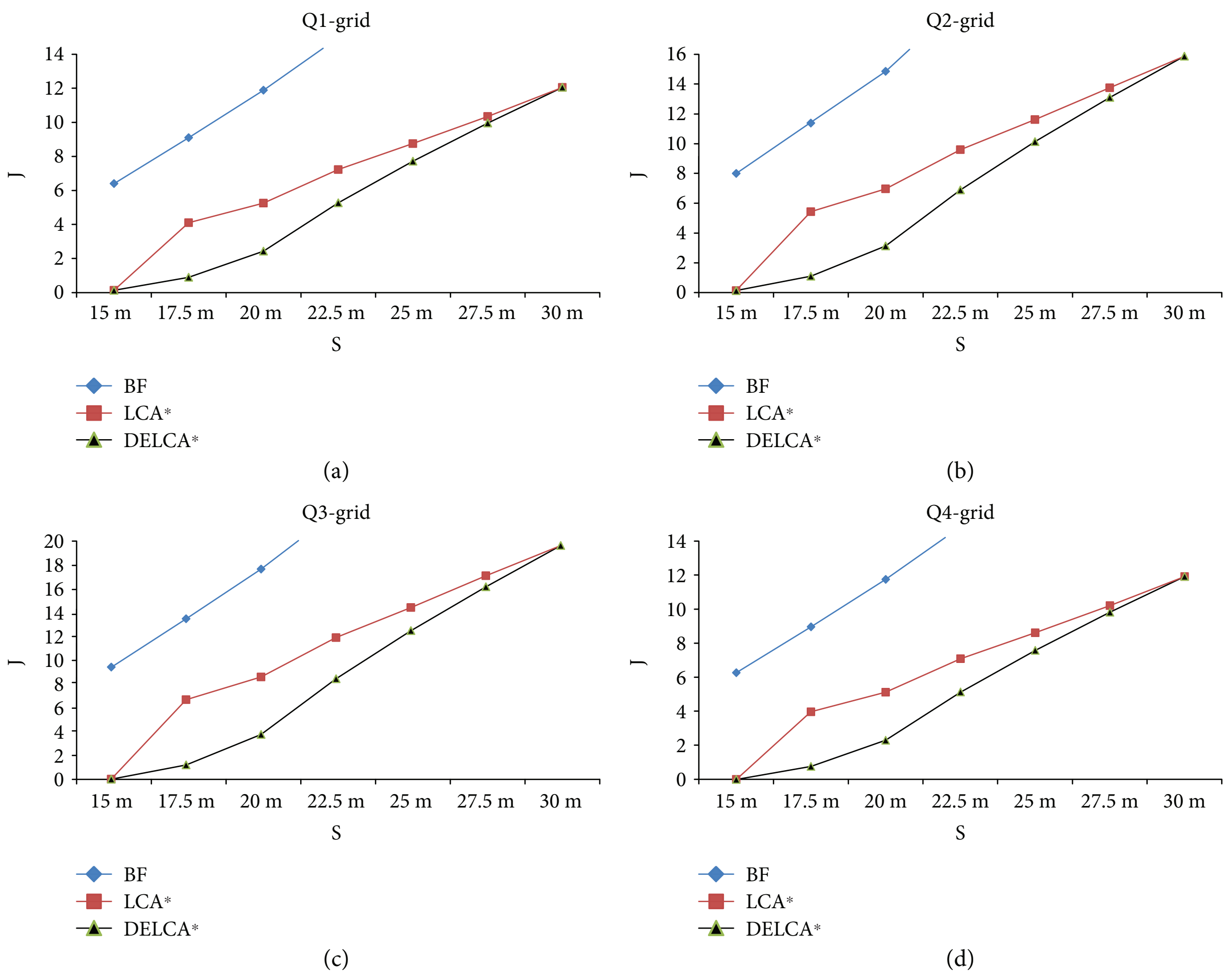


Figure 8: Energy consumption on grid placement with varying *s*.

animals which satisfy the selection predicate (i.e., height ≤ 200). Query Q3 requests the average body temperature of small sized animals. Query Q4 is a more complex nested query to obtain the position and body temperature of each animal with the highest body temperature. In the above queries, the default value of the minimum similarity threshold δ is 0.995. The default semantic policy for duplicate identification is MONID, and the size of a LSH vector is 16 bits.

5.2. Experimental Results

5.2.1. Varying the Sensing Radius (s). We varied the radius of sensing region s from 15 meters to 30 meters. Each sensor node detected the animals every minute, and we ran the simulator for 10 minutes. Figure 8 shows the energy consumption of each algorithm for the four test queries with the grid placement strategy of sensor nodes. Figure 9 shows the corresponding energy consumption based on the random placement strategy. As shown in Figures 8 and 9, BF shows the worst performance as we expected since each sensor sends its readings to the base station. In contrast, DELCA* shows the best performance in most cases since the partial aggregation results of the unique objects and potential duplicates are transmitted to the base station during the second phase. On average, with the grid placement strategy, the performance of DELCA* is 2.73 times and 1.25 times better than those of BF and LCA*, respectively. Note that as the sensing radius becomes larger, the performance gap between DELCA* and LCA* decreases, and LCA* even shows better performance than DELCA* with the largest sensing radius. The reason is that as the overlapped sensing region increases, the number of duplicate objects increases. In LCA*, each sensor node simply sends its detected objects to its coordinator since LCA* does not distinguish unique objects and potential duplicates. Meanwhile, in DELCA*, each sensor consumes its energy to identify unique objects from its detected objects by collaborating with its oneighbors during the first phase. Thus, the gain of identifying the unique objects is compensated with the overhead of the first phase. In those cases, LCA* can show better performance.

With the grid placement strategy, when the sensing radius is small (i.e., $s = 15$ meters), the performances of LCA* and DELCA* are the same since the sensing regions of the sensors are disjoint, and thus, each sensor node becomes the coordinator of itself and every detected object is unique. Meanwhile, with the random placement strategy, due to the irregularity of the sensor locations, there are overlapped sensing regions, and thus, the energy consumptions of

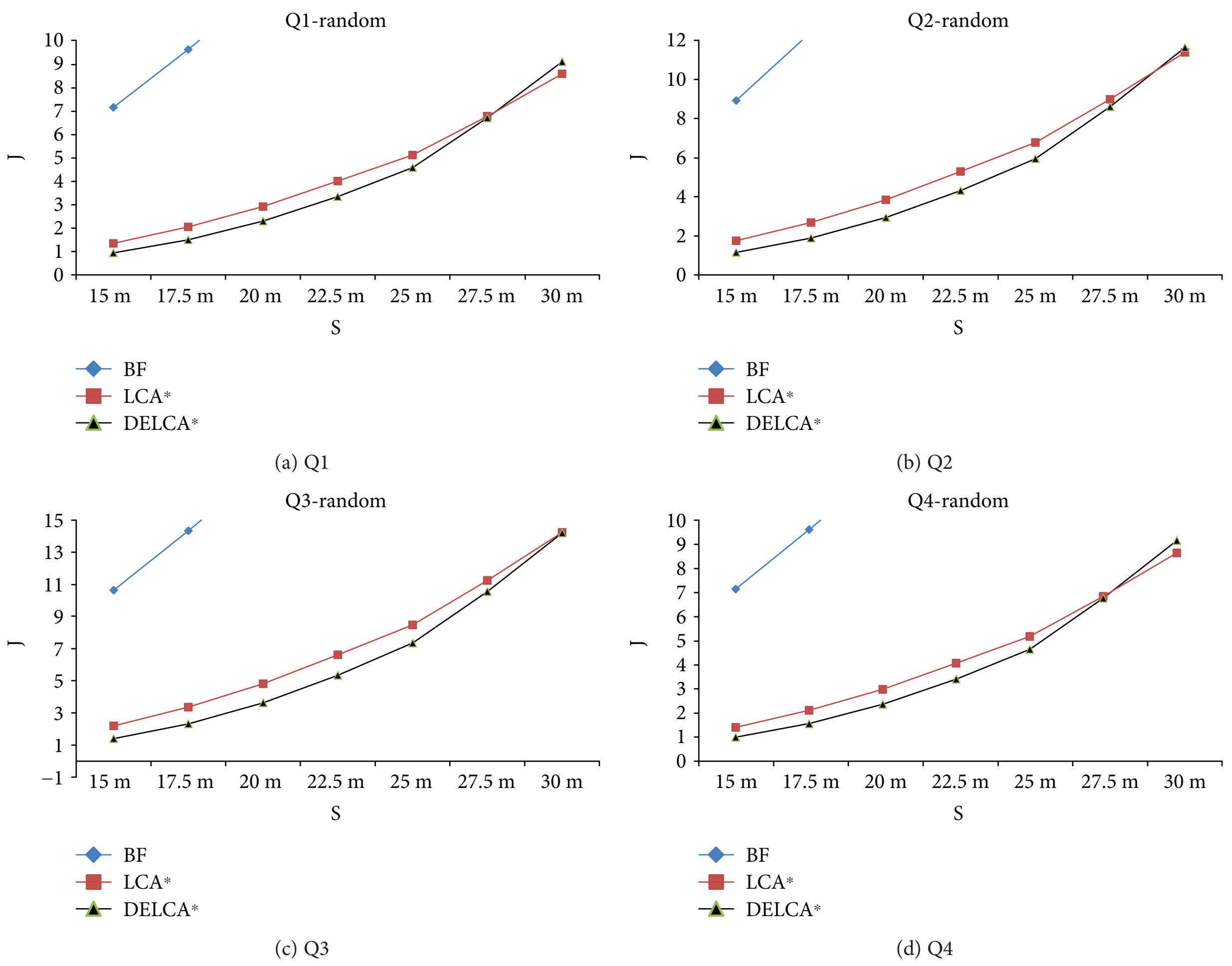


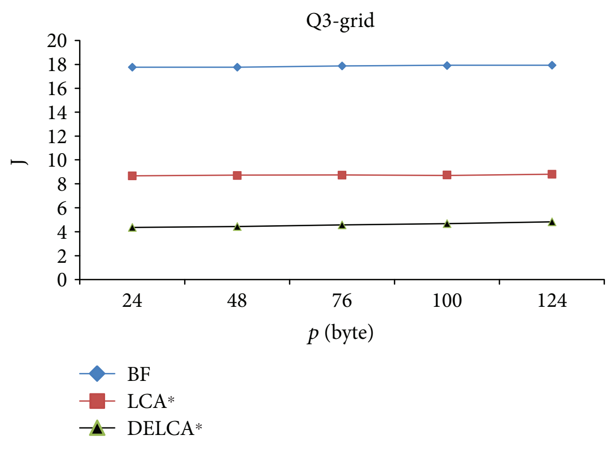
FIGURE 9: Energy consumption on random placement with varying *s*.

LCA* and DELCA are different from those with the grid placement strategy. In addition, due to such irregularity, since there are some nodes having no oneighbors and they are the coordinators of themselves, we do not need to perform the deduplication for those nodes. Furthermore, since LCA* does not perform the first phase of DELCA*, the performance gap between LCA* and DELCA* with the random placement strategy is less than that with the grid placement strategy. On average, DELCA* is 1.1 times better than LCA* when compared with the random placement strategy.

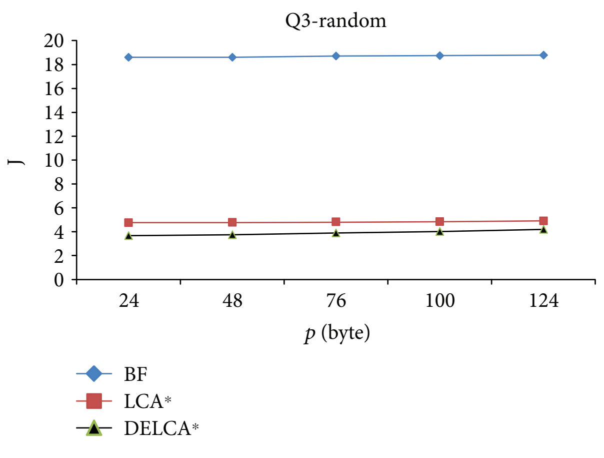
5.2.2. Varying the Packet Size (p). We also varied the packet size p from 24 bytes to 124 bytes. Figure 10 shows the results of Q3 with the grid and random placement strategies. We do not present the result of other queries since the performance patterns are very similar. As addressed in [7], the energy consumption of each algorithm is not much affected by the change of the packet size, since the number of packets to be transmitted is reduced but the energy to transmit a packet increases as the packet size p increases. The results plotted in Figure 10 also confirm this analysis.

5.2.3. Varying the Number of Objects (a). We next varied the number of object a from 1,000 to 20,000 with the default values of other parameters. In Figure 11, we plot only the experimental result of Q3 with the grid placement and random placement strategies since those of other queries show the similar pattern. As illustrated in Figure 11, the energy consumption of each scheme increases with growing the number of objects since each node detects more objects. Moreover, as the number of objects increases, the number of unique objects also increases with the fixed radius of sensing regions. Thus, BF shows the worst performance, and the performance gap between LCA* and DELCA* increases. On average, the performance of LCA* is 2.70 times better than that of BF. In addition, DELCA* is 1.61 times better than LCA*.

5.2.4. Relative Error. Finally, we report the accuracy of the tested algorithms with the three duplicate semantics (i.e., the weak, strict, and monoid semantics). To measure the accuracy, we compared with an ideal system in which each sensor node detects the objects accurately without any duplicate detection. We computed the relative error err of every algorithm with respect to the result of the ideal system such that $err = |Agg_{\text{ideal}} - Agg|/Agg_{\text{ideal}}$, where Agg_{ideal} is the aggregation result by the ideal system and Agg is that of each algorithm. In Figure 12, we plotted the relative errors of the aggregation results of query Q3 on the grid placement strategy with. We plot the relative errors of the aggregation results

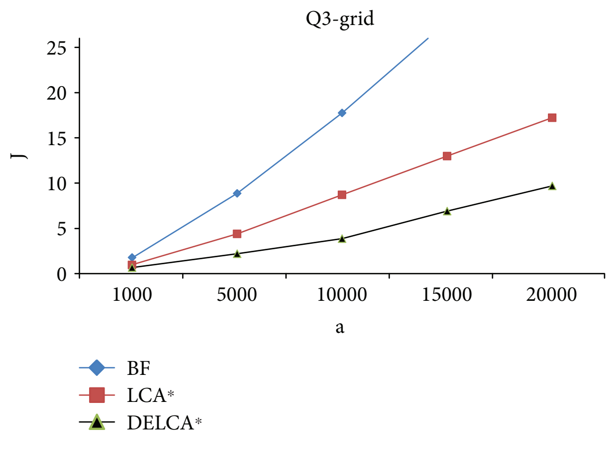


(a) Q3-grid

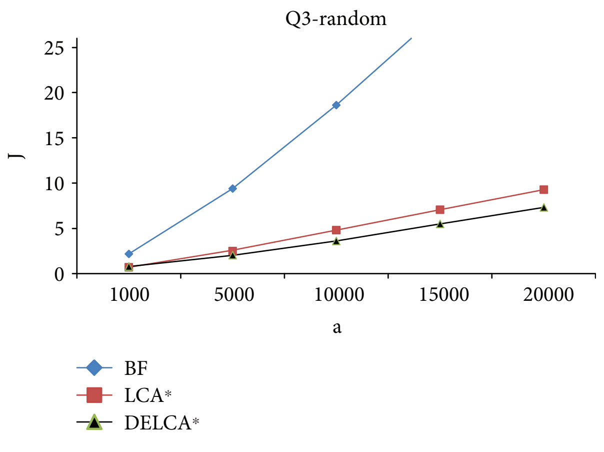


(b) Q3-random

FIGURE 10: Energy consumption with varying p.

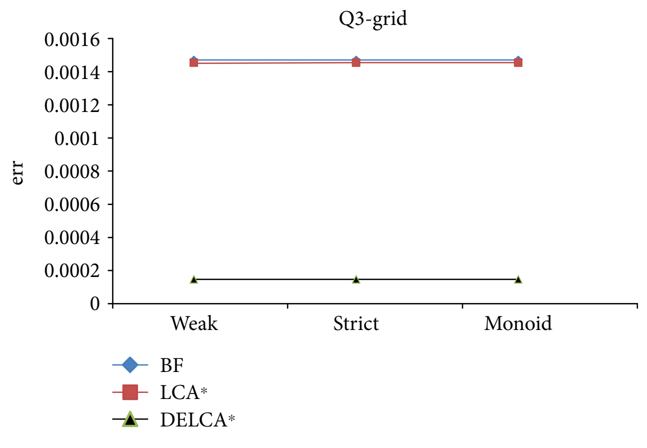


(a) Q3-grid

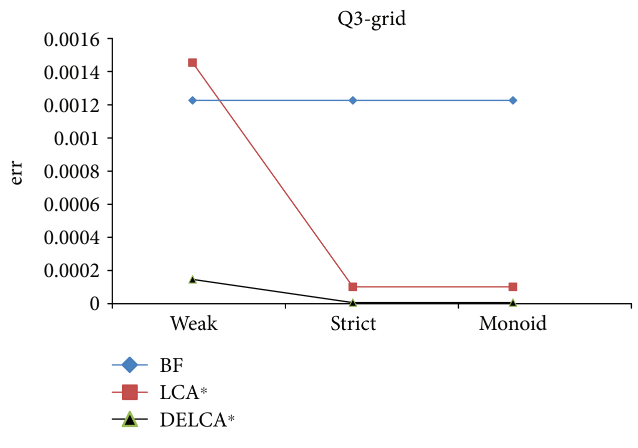


(b) Q3-random

FIGURE 11: Energy consumption with varying a.



(a) Q3-grid when δ is 0.995



(b) Q3-grid when δ is 0.998

FIGURE 12: Relative error.

of Query Q3 on the grid placement strategy with $\delta = 0.995$ and $\delta = 0.998$.

As shown in Figure 12, as δ increases, the relative error decreases. Furthermore, the relative error is much affected by each of the query processing methods rather than the duplicate semantics. In LCA* and BF methods, each coordinator (and the base station) determines the unique objects among the objects coming from its descendants irrespective of the overlaps of the sensing regions. Thus, even though two objects are detected by different sensor nodes whose sensing areas are disjoint, these objects can be regarded as the duplicates, if the similarity value is at least δ. Recall that each sensor node determines the unique objects at the end of the first phase by using LSH vectors in DELCA*. Since the unique objects are determined at each node by collaborating with its oneighbors only, DELCA* is more accurate than the other methods.

6. Conclusion

In this paper, we study the situation when sensors can have overlapping sensing regions, and the objects have no explicit identifiers. When the sensing regions of sensors can overlap, an object can be detected by several sensor nodes. Such detection of duplicates by several sensors makes the in-network query processing in WSNs problematic. In this paper, we propose an approximate but effective in-network aggregation algorithm, called DELCA*, to compute aggregation queries for the objects without any identifiers. We utilize the LSH to split the unique objects and potential duplicates in the first phase of DELCA*. Then, during the second phase, by performing deduplication, partial in-network aggregation can be conducted. In addition, to alleviate the nontransitivity issue of the similarity measures, we consider three duplicate semantics. To demonstrate the efficiency of our proposed algorithm DELCA*, we measure the transmission costs of three aggregation processing methods: brute-force (BF) method, DELCA*, and LCA*. In our experiments, BF shows the worst performance since every detected object is transmitted to the base station. Meanwhile, DELCA* shows the best performance in most cases since the partial aggregation result of the unique objects and potential duplicates are transmitted to the base station during the second phase. However, as the sensing radius becomes larger, the performance gap between DELCA* and LCA* decreases as the number of duplicated objects increases. In our experiments, the relative error is more significantly affected by the query processing methods used than the duplicate semantics. BF and LCA* determine the unique objects without considering the overlaps of the sensing regions. However, since each sensor node identifies unique objects among its detected objects by collaborating with its oneighhors, DELCA* is the most accurate. In a future work, we will investigate a technique that chooses DELCA* and LCA* adaptively by considering the network configuration, sensing radius, and the number of potential duplicates.

Data Availability

The Kruger Buffalos data used to support the findings of this study have been deposited in the Movebank Repository (http://www.movebank.org).

Conflicts of Interest

The authors declare that there is no conflict of interests regarding the publication of this article.

Acknowledgments

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2015 R1D1A1A01058909). It was also supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (NRF-2017M3C4A7063570).

References

[1] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin, “Habitat monitoring with sensor networks,” *Communications of the ACM*, vol. 47, no. 6, pp. 34–40, 2004.

[2] W. M. Merrill, F. Newberg, K. Sohrabi, W. Kaiser, and G. Pottie, “Collaborative networking requirements for unattended ground sensor systems,” in *IEEE Aerospace Conference Proceedings (Cat. No.03TH8652)*, Big Sky, MT, USA, March 2003.

[3] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, and Y. Yao, “The cougar project: a work-in-progress report,” *ACM SIGMOD Record*, vol. 32, no. 4, pp. 53–59, 2003.

[4] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong, “Model-driven data acquisition in sensor networks,” in *Proceedings 2004 VLDB Conference*, pp. 588–599, Toronto, Canada, 2004.

[5] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “Tag: a tiny aggregation service for ad-hoc sensor networks,” in *Proceedings of the 5th symposium on Operating systems design and implementation - OSDI '02*, pp. 131–146, Boston, Massachusetts, December 2002.

[6] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “The design of an acquisitional query processor for sensor networks,” in *SIGMOD '03 Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pp. 491–502, San Diego, CA, USA, June 2003.

[7] J.-K. Min, R. T. Ng, and K. Shim, “Aggregate query processing in the presence of duplicates in wireless sensor networks,” *Information Sciences*, vol. 297, pp. 1–20, 2015.

[8] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, “Energy-efficient computing for wildlife tracking,” *ACM SIGARCH Computer Architecture News*, vol. 30, no. 5, pp. 96–107, 2002.

[9] P. Indyk and R. Motwani, “Approximate nearest neighbors,” in *STOC '98 Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613, Dallas, TX, USA, 1998.

[10] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “Tinydb: an acquisitional query processing system for sensor networks,” *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 122–173, 2005.

[11] S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson, “Synopsis diffusion for robust aggregation in sensor networks,” *ACM Transactions on Sensor Networks*, vol. 4, no. 2, pp. 1–40, 2008.

[12] A. Gionis, P. Indyk, and R. Motwani, “Similarity search in high dimensions via hashing,” in *Proceedings of the 25th VLDB Conference*, pp. 518–529, Edinburgh, Scotland, 1999.

[13] M. S. Charikar, “Similarity estimation techniques from rounding algorithms,” in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing - STOC '02*, pp. 380–388, Montreal, Quebec, Canada, May 2002.

[14] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, “Locality-sensitive hashing scheme based on p-stable distributions,” in *Proceedings of the twentieth annual symposium on Computational geometry - SCG '04*, pp. 253–262, Brooklyn, NY, USA, June 2004.

[15] J. Considine, F. Li, G. Kollios, and J. Byers, “Approximate aggregation techniques for sensor databases,” in *Proceedings 20th International Conference on Data Engineering*, pp. 449–460, Boston, MA, USA, April 2004.

[16] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, “Medians and beyond: new aggregation techniques for sensor networks,” in *Proceedings of the 2nd international conference on Embedded networked sensor systems - SenSys '04*, pp. 239–249, Baltimore, MD, USA, November 2004.

[17] A. Silberstein, K. Munagala, and J. Yang, “Energy-efficient monitoring of extreme values in sensor networks,” in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data - SIGMOD '06*, pp. 169–180, Chicago, IL, USA, June 2006.

[18] S. Xiao, B. Li, and X. Yuan, “Maximizing precision for energy-efficient data aggregation in wireless sensor networks with lossy links,” *Ad Hoc Networks*, vol. 26, pp. 103–113, 2015.

[19] X. Xu, R. Ansari, and A. Khokhar, “Power-efficient hierarchical data aggregation using compressive sensing in WSNs,” in *IEEE International Conference on Communications (ICC)*, pp. 1769–1773, Budapest, Hungary, June 2013.

[20] D.-W. Choi and C.-W. Chung, “Request: region-based query processing in sensor networks,” in *Database Systems for Advanced Applications. DASFAA 2011. Lecture Notes in Computer Science, vol 6588*, J. X. Yu, M. H. Kim, and R. Unland, Eds., pp. 266–279, Springer, Berlin, Heidelberg, 2011.

[21] A. Soheili, V. Kalogeraki, and D. Gunopulos, “Spatial queries in sensor networks,” in *Proceedings of the 2005 international workshop on geographic information systems - GIS '05*, pp. 61–70, Bremen, Germany, November 2005.

[22] Y. Zhuang and L. Chen, “Max regional aggregate over sensor networks,” in *2009 IEEE 25th International Conference on Data Engineering*, pp. 1295–1298, Shanghai, China, March April 2009.

[23] G. Mali and S. Misra, “Topology management-based distributed camera actuation in wireless multimedia sensor networks,” *ACM Transactions on Autonomous and Adaptive Systems*, vol. 12, no. 1, pp. 1–33, 2017.

[24] C.-W. Ngo, W.-L. Zhao, and Y.-G. Jiang, “Fast tracking of near-duplicate keyframes in broadcast domain with transitivity propagation,” in *Proceedings of the 14th annual ACM international conference on Multimedia - MULTIMEDIA '06*, pp. 845–854, Santa Barbara, CA, USA, October 2006.

[25] C. Gong, Y. Huang, X. Cheng, and S. Bai, “Detecting near-duplicates in large-scale short text databases,” in *Advances in Knowledge Discovery and Data Mining. PAKDD 2008. Lecture Notes in Computer Science, vol 5012*, T. Washio, E. Suzuki, K. M. Ting, and A. Inokuchi, Eds., pp. 877–883, Springer, Berlin, Heidelberg, 2008.

[26] F. Xu and C. Jermaine, “Randomized algorithms for data reconciliation in wide area aggregate query processing,” in *VLDB '07 Proceedings of the 33rd international conference on Very large data bases*, pp. 639–650, Vienna, Austria, September 2007.

[27] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, “An application-specific protocol architecture for wireless microsensor networks,” *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, pp. 660–670, 2002.

[28] “MoveBank,” http://www.movebank.org.